AD-A054 900 POLYTECHNIC INST OF NEW YORK FARMINGDALE DEPT OF ELE--ETC F/G 9/2
SOFTWARE MODELING STUDIES. VOLUME II. STATISTICAL (NATURAL) LAN--ETC(U)
APR 78 A LAEMMEL, M SHOOMAN F30602-74-C-0294

UNCLASSIFIED RADC-TR-78-4-VOL-2 NL

1 OF 1
AD
A054900

END
DATE
FILMED
7-78
DDC

FOR FURTHER TRAN

A052615

ADA054900

RADC-TR-78-4, Vol II (of two)
Final Technical Report
April 1978

(2)

SOFTWARE MODELING STUDIES
Statistical (Natural) Language Theory and Computer Program Complexity

Polytechnic Institute of New York

A. Laemmel
M. Shooman

Approved for public release; distribution unlimited

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

RADC-TR-78-4, Vol II (of two) has been reviewed and is approved for publication.

APPROVED: Alan N. Sukert
ALAN N. SUKERT
Project Engineer

APPROVED: Wendall C Bauman
WENDALL C. BAUMAN, Col, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: [illegible]
JOHN P. HUSS
Acting Chief, Plans Office

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 **REPORT DOCUMENTATION PAGE**		READ INSTRUCTIONS BEFORE COMPLETING FORM
18 1. REPORT NUMBER RADC-TR-78-4, VOL [redacted] -2	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) 6 SOFTWARE MODELING STUDIES. Volume II. Statistical (Natural) Language Theory and Computer Program Complexity.		5. TYPE OF REPORT & PERIOD COVERED 9 Final Technical Report. 1 Apr[redacted]74 — 1 Oct[redacted]77
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(s) 10 A. Laemmel M. Shooman		8. CONTRACT OR GRANT NUMBER(s) 15 F30602-74-C-0294
9. PERFORMING ORGANIZATION NAME AND ADDRESS Polytechnic Institute of New York 333 Jay Street Electrical Eng & Electrophysics Brooklyn NY 11201		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62728F 16 55500806 17 08
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIS) Griffiss AFB NY 13441		12. REPORT DATE 11 Apr[redacted]78
		13. NUMBER OF PAGES 48
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same 12 56p.		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release: distribution unlimited.

1- A052 615

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Alan N. Sukert (ISIS)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Natural Language Theory
Zipf's Laws
Computer Program Complexity
Computer Programming Language Complexity

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report discusses the application of concepts of statistical language theory (Zipf's Laws) to the derivation of formulas for measuring program and language complexity. Experimental data from several different programs and programming languages, such as PL/I, assembly and FORTRAN, is presented which is used to verify the necessary underlying assumption and to verify formulas for program length by comparison with actual statistics. Finally, the derived formulas are compared with those of Software Physics derived by Halstead.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

409 203

mt

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF FIGURES (continued)

LIST OF TABLES

EVALUATION

The necessity for producing more reliable, lower cost software for such military applications as command and control, and avionics has led to the desire to develop newer tools, techniques, and predictive aids for improving the method in which software is currently being developed and tested. This desire has been expressed in such documents as the Joint Logistics Commanders' Software Reliability Working Group Report (November 1975) and in numerous industry and Government sponsored conferences and symposia. As a result, efforts have begun to develop and test mathematical models for predicting the error content of a software package as well as determining test criteria and measures of complexity. However, early efforts to develop such models have been fragmented and have not produced models with the desired accuracy and useablility.

This effort was initiated in response to this need for developing accurate software predictive models and fits into the goals of RADC TPO No. 5, Software Cost Reduction (formerly RADC TPO No. 11, Software Sciences Technology), in particular the area of Software Quality (Software Modeling). This report summarizes the development of measures of language and program complexity based upon the application of natural language theory, and in particular Zipf's laws. The importance of this development is that it represents the first cohesive attempt to apply natural language theory to the development of software complexity measures, which in turn will produce a comprehensive theory of software complexity that will permit prediction of development effort and error content.

The theory developed under this effort will lead to much needed tools for use by software managers in adequately tracking the progress of a software development in terms of the size of effort or number of errors. In addition, the theory developed under this effort will provide the basis for further development that will eventually produce the capability to completely define and measure software complexity. Finally, the measures developed under this effort will be applicable to current Air Force software development projects and thus help to produce the high quality, low cost software needed for today's systems.

Alan N. Sukert
ALAN N. SUKERT
Project Engineer

ABSTRACT

There is a great need for measures of program complexity. Such measures find applications in (1) early design estimates of program difficulty; (2) comparison measures of programs; and (3) as a normalizing parameter in the analysis of experimental data on errors and man-hours. We show that using established concepts of statistical language theory (Zipf's law) one is able to derive formulas for program complexity similar to those obtained by Halstead[17] in his work on Software Physics. The technique can be used to measure and study language complexity by applying it to an actual program and comparing the calculated complexity with a theoretical minimum problem-program complexity.

We present experimental data on PDP-11 assembly language, PL/I programs, M6800 assembly language, and FORTRAN statements, which generally verify the necessary underlying assumption (Zipf's law holds for computer languages). Applying Zipf's law we obtain formulas for program length which can be verified by comparison with statistics obtained from computer programs.

The formulas are compared with Halstead's work and ours. Both sets of length formulas agree within 10%-20% of the actual length (operators plus operands) although neither is better for all examples.

1.0 Introduction

There is a great need for theoretical models which describe programs and allow us to quantitatively estimate complexity, running time, storage requirements, and development time. In addition to serving as an estimate during the initial design period, they can be refined as the program develops and used as a management and analysis tool. They can also be used to compare initial design approaches, programming styles, different algorithms, etc. Early work on such a theory is given in Refs. 19 and 20.

This report is closely linked to Ref. 19 which develops a theory of program complexity based on linguistic theory, information theory and communications theory. This work discusses the background linguistic theory (Zipf's laws), extends this theory to programming languages, and develops equations for program length based on the operators, operands, and constants in the program. The results are different from those in Halstead's work on <u>Software Physics</u>[17]; however, in many examples they are numerically close. The main differences lie in the basic assumptions and theoretical development.

2.0 Zipf's Laws of Natural Language

<u>2.1 Introduction</u>

Any investigator who has studied ways of describing the structure of computer languages comes to the immediate conclusion that the task is a formidable one due to the seemingly infinite variety of programs we can write with a computer language. Much can be learned in tackling this problem by studying the analogous and even more complex problem of describing the structure of a written or spoken natural language. A set of analogies between the elements of natural and computer languages is given in Table 1.

Table 1 - Analogies between elements of natural and computer languages.

Natural Literature (or dialog)	Computer Programs (software)
Book or Speech	Software System
Chapter, Article, Conversation	Program
Paragraph, Section	Subprogram, Module, Procedure
Sentence	Statement
phrase, clause	expression
subject	
predicate	
Word	Element between spaces
noun	operand
adjective, adverb	?
verb	operator
auxilary verb	?
preposition, conjunction	?
pronoun	?
article	?
Punctuation	Special characters
period	semicolon (PL/I),CR,etc.

Inspection of Table 1 shows a parallelism of structure between natural and computer languages. For example, nouns correspond to operands, and punctuation marks parallel special characters. An exact analogy exists if we describe the natural language using Chomsky grammar[22] and the computer language using BNF form[23], which is essentially an adaptation of Chomsky's work. However, it will only be necessary to consider words, statements, operators, and operands in the presentation of data and derivation of formulas which follows, and each of these is easy to define and deal with.

The reader may comment that natural and computer languages are really different things and like apples and oranges cannot be compared. The following thought problem rebutes this argument. Suppose we take a programmer who understands a particular computer language and give him a complete listing with comments and documentation of a computer program. We instruct him to study the computer program until he understands it and then produce a report written in good English containing paragraphs, complete sentences, and algorithms written as a sequence of steps without mathematical notation. This would produce a report which another programmer could read and hand process (perhaps with the aid of a nonprogrammable calculator) to produce an output. In principle, the report and the computer program would be equivalent. Thus, any studies one might make of the elements of the computer program would have to bear a close resemblance to the equivalent elements of the report.

2.2 Zipf's First Law

Before we discuss Zipf's law it is convenient to introduce a few terms used in dealing with natural language. Unfortunately, the term word is ambiguous in the sense that all the words in the abstract of this report mean something different than all the words in the dictionary. We use the term token to refer to all the words of the abstract. The term type is used to refer to the words in the dictionary. Much of our effort will be centered on the counting of n_r, the number of times the r^{th} particular type occurs in a sample of n tokens. We have used the subscript r because in our comparison of word types we will order the types in terms of their frequency and assign ranks. The most frequently occurring type will be assigned rank 1, the second most frequent type rank 2, etc. If we assume there are t word types in our sample of n tokens then clearly

$$\sum_{r=1}^{t} n_r = n \tag{1}$$

Clearly, when we begin to collect our statistics there will be a number of types, t_k, which occur the same number of times, k. These will essentially tie for the same rank r, so we can arbitrarily assign the ranks from r to $r-k+1$ to these types which occur k times. Using the above definitions we can write a summation equation similar to Eq. 1 for the types

$$\sum_{k=1}^{n} t_k = t \tag{2}$$

The absolute frequency of occurrence for type r is n_r; however the relative frequency of occurrence f_r is simply n_r/n.

Zipf studied the relationship between frequency of occurrence n_r and rank r for words from English, Chinese, and the Latin of Platus.[1] The resulting data is given in Fig. 1, where Zipf plotted the $\log_{10}$ of n_r on the abcissa and the $\log_{10}$ of r on the ordinate. Note that the data for English is an excellent fit to a straight line, as is the data for Latin for ranks larger than 10.

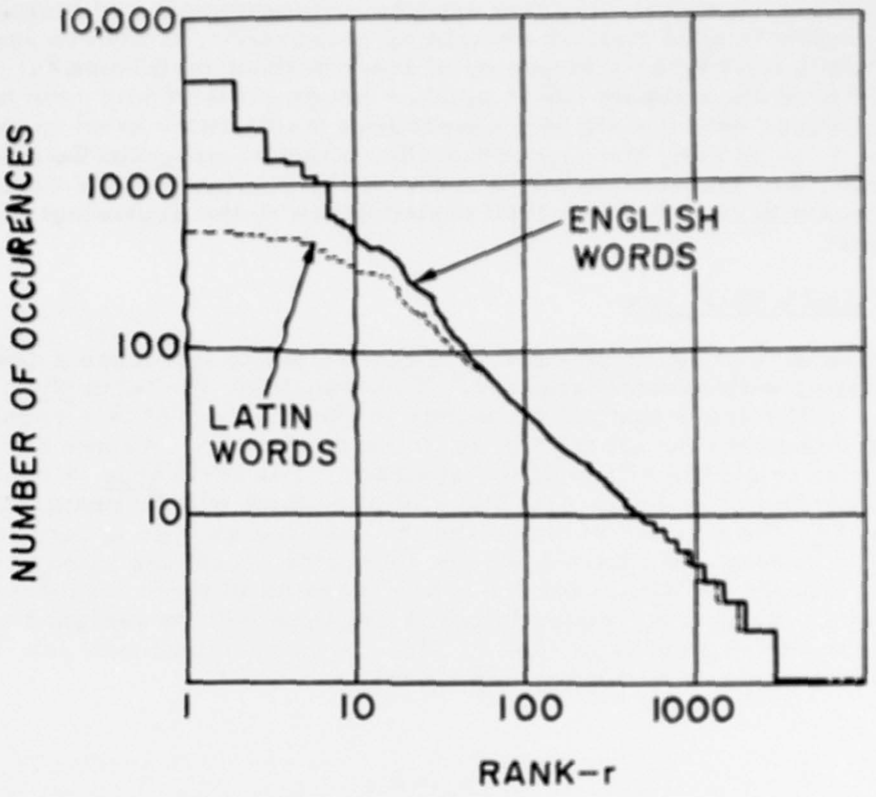

FIG. 1 OCCURRENCE FREQUENCY VS. RANK FOR ENGLISH AND LATIN WORDS. (FROM ZIPF, REF. 1, PLATE IV).

If we had plotted $\log_{10} f_r$ on the abcissa we would have obtained the same result shifted by the scale factor n. The straightness of such plots on log-log paper is an empirical justification (we will discuss later theoretical justifications) for the relationship

$$f_r \cdot r^a = \text{Constant} = c \quad . \tag{3}$$

Careful study of Zipf's data and that of others shows the constant a, (the slope on log-log paper) is approximately 1: thus we arrive at the simple relationship, generally called Zipf's first law, given below

$$f_r \cdot r = c \tag{4}$$

which can also be written in the form

$$n_r = \frac{c \cdot n}{r} \tag{5}$$

Inspection of Eq. 4 yields the fact that the constant c can be interpreted as the relative frequency of the rank 1 word type. There are some theoretical and practical problems with this obvious interpretation. In a practical sense, if we use the ratio of n_1/n to determine the constant c, we will probably obtain a poorer estimate of c than could be obtained by measuring the y-axis intercept of the straight line fitted to the data on log-log paper, since Zipf's law holds less well at times for the lowest rank. Theoretically, we can examine some of the properties of the frequency distribution to see if we can interpret it as a probability distribution. Letting p_r be the probability of occurrence for the rth ranked word, the summation of all the p_r's yields

$$\sum_{r=1}^{t} p_r = \sum_{r=1}^{t} f_r = \sum_{r=1}^{t} \frac{n_r}{n} \tag{6}$$

Substituting Eq. 1 into Eq. 6 we obtain

$$\sum_{r=1}^{t} p_r = 1 \tag{7}$$

as of course it should.

2.3 Word Type

We can obtain other fundamental relations based on the number of types via summation of Zipf's law. If we sum both sides of Eq. 5 we get a different result.

$$\sum_{r=1}^{t} n_r = c\,n \sum_{r=1}^{t} \frac{1}{r} \tag{8}$$

The summation of the series 1/r is given by

$$\sum_{r=1}^{t} \frac{1}{r} = 0.5772 + \ln t + \frac{1}{2t} - \frac{1}{12t(t+1)} \cdots \qquad (9)^{*}$$

Substitution from Eq. 1 and Eq. 9 (retaining only 2 terms for modest size t) into Eq. 8, and rearrangement yields an expression for the constant c in terms of t.

$$c = \frac{1}{0.5772 + \ln\ t} \tag{10}$$

* L. Jolley, "Summation of Series," p. 36, n. 200, and p. 14, n. 70, Dover Publications, NY 1961. (Note the constant 0.5772 is called Eulers constant, see "Differential and Integral Calculus," R. Courant, vol. 1, Interscience Publishers, NY 1951, p. 381, 420.)

If we know the number of types, t. we can use Eq. 10 to estimate c. However, we notice that as t increases, c decreases. From Eq. 4 $f_1 = c$; thus, the frequency of the first rank (and all the other ranks) changes with c which in turn varies with t. However, if we think of our statistics as a sample and apply the relative frequency interpretation of probability, we would expect f_r to approach finite values as the sample size (number of tokens, n) approached infinity. It appears that in natural language text, we would expect t to increase with n. The concept of p_1 being a function of sample size is contradictory if we assume random statistically independent tokens or the n tokens represent an arbitrary slice from a larger work. However, if we require that the n tokens represent some cohesive unit (e.g., abstract, paragraph, chapter, paper, etc.) then it is reasonable that the author's style and work probabilities will change with length.* If such is the case we would need data for different token lengths and their associated values of c so that we would investigate the validity of Eq. 10. The way in which t varies with sample size n is stated by examining how the type-token ratio, $\tau = t/n$, varies with n.

We can derive an equation for the type-token ratio by considering the behavior of Eq. 5 for the smallest rank, which is where $r_{max} = t$ (e.g., if there are 100 types, then the largest rank is obviously 100). In most cases the rarest type (largest rank) will occur only once; thus, $n_{r_{max}} = 1$. Substituting these values in Eq. 5 yields another interpretation for the constant c

$$\tau \equiv \frac{t}{n}\ c \quad . \tag{11}$$

Equating Eq. 10 to Eq. 11 yields

$$\tau \equiv \frac{1}{0.5772 + \ln t} \tag{12}$$

and we now have an expression for the dependence of τ on t. In order to relate τ to n we substitute $t = n\tau$ in Eq. 12 and solve for n obtaining

$$n = \frac{1}{\tau} e^{\left(\frac{1}{\tau} - 0.5772\right)} \tag{13}$$

Clearly, n increases with decreasing τ or equivalently τ decreases with n. The shape of this relationship between τ and n agrees well with the shape of the sparse data in the literature; however, the numerical values are in poor agreement.** We can obtain an equation relating n to t by substituting t/n for τ in Eq. 12 and solving

$$n = t(0.5772 + \ln t) \quad . \tag{14}$$

* cf. Ref. 15 p. 245.

** See Miller Ref. 4, p. 124.

2.4 Zipf's Second Law

We can derive a second form of Zipf's law from the first form by making other assumptions about the behavior in the region where r is larger.[13, 16] We compute the derivative of n_r with respect to r from Eq. 5 and take its magnitude

$$\left|\frac{dn_r}{dr}\right| = \left|\frac{-nc}{r^2}\right| = \frac{cn}{r^2} \quad . \tag{15}$$

Eliminating r by substitution from Eq. 5 into Eq. 15 yields

$$\left|\frac{dn_r}{dr}\right| = \frac{n_r^2}{cn} \quad . \tag{16}$$

In the tail of Zipf's law, there are several identical n_r values; thus there is a plateau of k types with the same n_r value which are tied for the same rank. The curve becomes a staircase function and we define the slope as the vertical decrement (r to $r + 1 \equiv 1$) over the horizontal width (which is t_k), thus

$$\left|\frac{dn_r}{dk}\right| = \frac{1}{t_k} \quad . \tag{17}$$

In the tail $n_{r_{max}} = 1$ and $k = 1$, $n_{r_{(max-1)}} = 2$ and $k = 2$, thus in this region $n_r = k$, and substituting into Eqs. 17 and 16 yields the second form of Zipf's law,

$$t_k = \frac{cn}{k^2} \quad . \tag{18}$$

Experimental verification of Zipf's second law is given in Figs. 2, 3, and 4. We can also derive a second set of basic equations using the second law. Combining Eq. 2 with 18 we obtain

$$t = \sum_{k=1}^{n} \frac{cn}{k^2} \approx \sum_{k=1}^{\infty} \frac{cn}{k^2} = \frac{\pi^2}{6} cn \tag{19}^*$$

*"Handbook of Mathematical Functions," NBS, 1964, p. 807, see *Riemann Zeta function.*

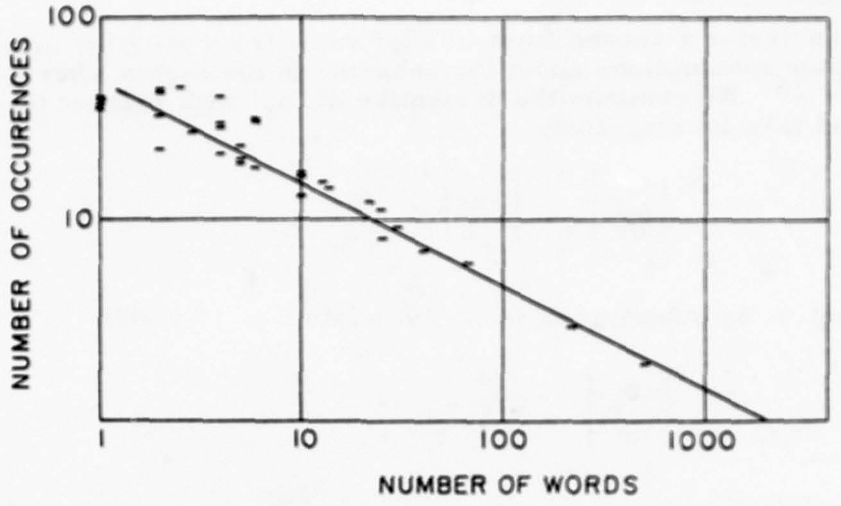

FIG. 2. EXPERIMENTAL VERIFICATION OF ZIPF'S SECOND LAW. (Chinese Words)

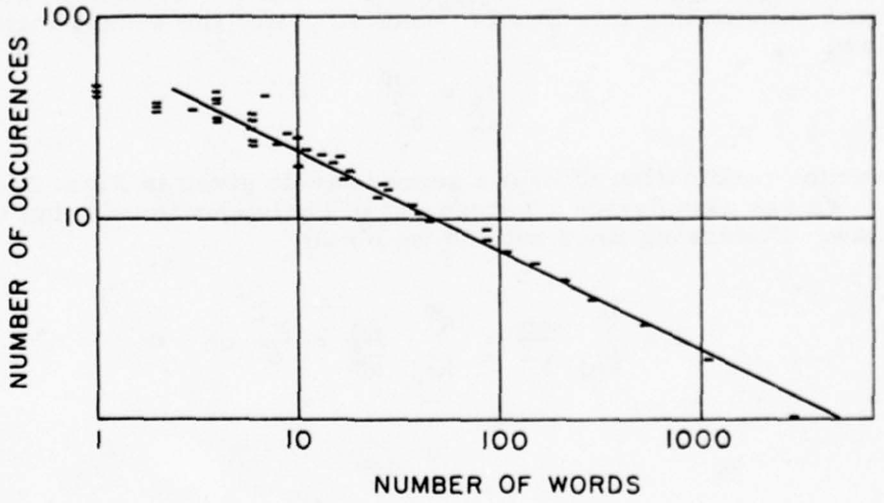

FIG. 3. EXPERIMENTAL VERIFICATION OF ZIPF'S SECOND LAW. (English Words)

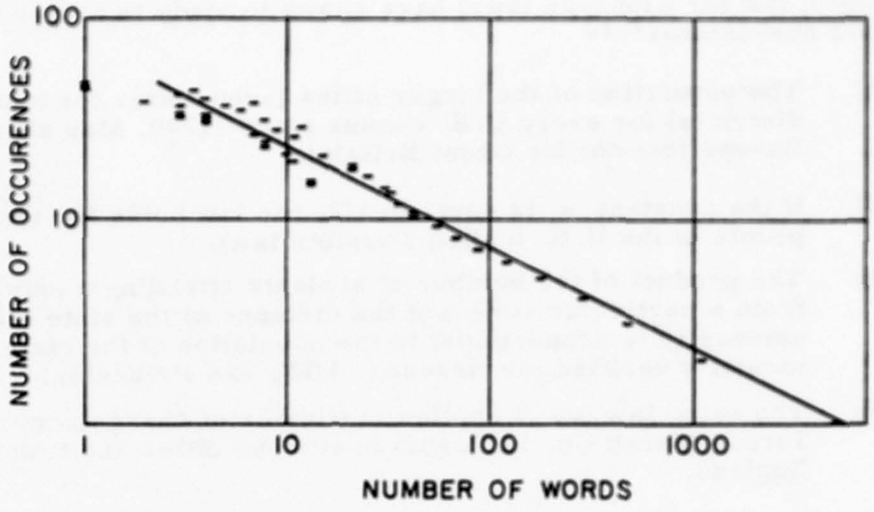

FIG. 4. EXPERIMENTAL VERIFICATION OF ZIPF'S SECOND LAW.
(Latin Words)

Combining Eq. 19 and 10 yields

$$n = \frac{6}{\pi^2}(0.5772 + \ln t)\, t \tag{20}$$

and by substituting $t = n\tau$ in Eq. 20 and solving we obtain

$$n = \frac{1}{\tau} e^{\left(\frac{\pi^2}{6\tau} - 0.5772\right)} \quad . \tag{21}$$

Note that except for a factor of $\pi^2/6$ the results given in Eqs. 13 and 21 and 14 and 20 are identical. The occurrence of either $6/\pi^2 = 0.6079$ or unity as a coefficient in these equations arises due to the slightly different assumptions made concerning the behavior of Zipf's law for small ranks in the two derivations.

Since Zipf's second law is derived from the first, we can verify the fact that a is unity (cf. Eq. 3) by plotting f_r vs. f(cf. Eq. 4) on log-log paper or t_k vs. k^2 (cf. Eq. 19) and checking for a unity slope. The z-axis intercept is the value of the constant c in the first law plot, and the intercept of the second law plot is cn. From Eq. 17 we see that the second law contains the derivative of the first law; thus, the statistical fluctuations of the data will be accentuated in the second law plot. (Thus the first law is better for curve fitting.)

Zipf's law (or Zipf type laws) have shown to apply to a wide variety of different phenomena:[7,12]

(1) The population of the larger cities (even better for metropolitan districts) for every U. S. census since 1790, also all cities of Europe (but not for Great Britain).

(2) If the constant a is equal to 1/2, the law holds for the income of people in the U. S. (called Pareto's law).

(3) The product of the number of students attending a university from a particular state and the distance of the state from the university is proportional to the population of the state (experimentally verified for Harvard, MIT, and Princeton).

(4) The same law (as 3) applies to number of charge accounts at Jordan Marsh Co. in Boston in various cities and towns of New England.

(5) The "interchange" between the cities is proportional to the product of the populations divided by the distance between them and applies to Telephone Calls, Railway Express packages, Truck and Passenger trips, and many other things.

(6) Many different languages and language elements.

Thus, Zipf's law seems to hold for a wide variety of "organized" behavior, of both humans and machines.

2.5 Generalized Zipf's Law

The fact that Zipf's law in its simple form fits many sets of data leads to the rather simple and powerful theory developed in Sections 2.2-2.4. In many cases, the experimental fit to Zipf's first law is only close enough for the ensuing theory to be thought of as a gross approximation. In such cases, two generalized forms of Zipf's laws will yield a better fit between theory and data. In such cases, the algebraic expressions and computations become more cumbersome. However, a simple computer program solves the computational problem and parametric studies provide sensitivity information on how the results change as parameters vary. Of course the simple Zipf theory will still hold well enough to provide some insight into the results.

One generalized form of Zipf's law (suggested by several authors) is of the form

$$p_r = \frac{n_r}{n} = \frac{C}{(r + A)^B} \tag{22}$$

We can easily see the role which the constants play by taking logarithms of both sides of Eq. 22 and rearranging them.

$$\ln p_r = \ln C - B \ln (r+A) \quad (23)$$

clearly if A= 0 and B = 1 we obtain Zipf's first law. The constant B represents the slope of the plot and A is a "shift" parameter. For convenience we will call this the slope-rank shift form. The role of A can be illustrated if we set B = 1 and r = 1

$$\ln p_1 = \ln C - \ln (1+A) = \ln \left(\frac{C}{1+A}\right) \quad (24)$$

If A=0, then C is the frequency of the first rank, i.e., the y axis intercept. If A=0.1 the intercept is about 1.1C and if A=-0.1 the intercept is about 0.9C. Examination of Eq. 23 shows that for large r, the effect of A is negligible. These results are illustrated in Fig. 5, where we see that B allows us to match a general slope other than unity and A adjusts for

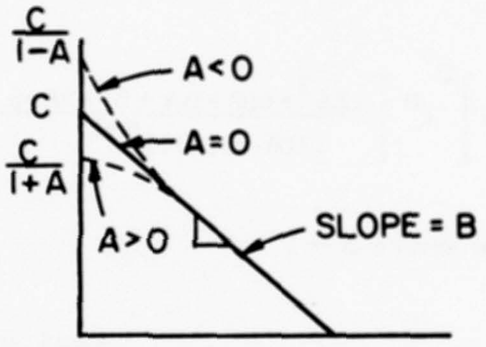

FIG. 5. ROLE OF COEFFICIENTS A, B, C IN EQ. 23.

derivations from straight line behavior for the initial ranks.

The physical meaning of the constants A, B, C have been studied in linguistics[4] and the following observations have been noted:

(1) Values of B greater than unity occur in individuals who possess a limited vocabulary, such as young children.

(2) Values of B less than unity are associated with large vocabularies, e.g., James Joyce or collections of writing by various authors as in a newspaper or anthology.

(3) Values of A greater than zero occur if the probability of the first rank word is small as is the case in English of words following "the", or in a highly inflected language.

(4) Values of A less than zero occur if the probability of the first word is large, as in the case of words following "of" ("the" being very frequent here).

In order to derive the modified basic equations for the slope-rank shift form of the law, we will apply Eq. 1 to Eq. 22

$$n = \sum_{r=1}^{t} n_r = Cn \sum_{r=1}^{t} \frac{1}{(r+A)^B} \qquad (25)$$

we can sum the above series with the aid of the Euler-Maclaurin Formula* which relates finite sum of a function to the integral of the function.

$$n = \left[\frac{12}{B^2+7B+12}\right]^B t^B \left[\frac{12A^2+6(B+3)A+B^2+5B+6}{12(B-1)(A+1)^{B+1}} - \frac{1}{(B-1)(t+A)^{B+1}}\right] \qquad (26)$$

For the special case where $B \to 1$

$$n \to \frac{3}{5} t \left[\ln(t+A) - \ln(A+1) + \frac{6A+7}{12(A+1)^2}\right] \qquad (27)$$

and where $A = 0$

$$n = \left[\frac{12}{B^2+7B+12}\right]^B t^B \left[\frac{B^2+5B+6}{12(B-1)} - \frac{1}{(B-1)t^{B-1}}\right] \qquad (28)$$

Similarly one can show that

$$cn = \left[\frac{12t}{B^2+7B+12}\right]^B \qquad (29)$$

* See Ref. 14, p. 322

$$t_k = \frac{(cn)^{1/B}}{B\,k^{(1+1/B)}} \qquad (30)$$

$$t = \frac{(B^2 + 7B + 12)(cn)^{1/B}}{12} \qquad (31)$$

$$\frac{t_k}{t} = \frac{12}{(B^2 + 7B + 12)\,k^{(1+1/B)}} \qquad (32)$$

When we plot Zipf law data, sometimes we have found experimental results on log-log paper which look like the straight lines with different slopes. (The reader who is familiar with the control system literature will notice the analogy with Bode plots.) Limiting our model to two piecewise-linear segments (see Fig. 6) we obtain

$$n_r = \left\{ \begin{array}{ll} \dfrac{C_1}{(r)^{B_1}} & 1 \leq r \leq D \\ \\ \dfrac{C_2}{(r)^{B_2}} & D \leq r \leq t \end{array} \right\} \quad \frac{C_1}{(D)^{B_1}} = \frac{C_2}{(D)^{B_2}} \qquad (33)$$

Again, by using the low order terms in the Euler-Maclaurin formula, on the two segments

$$n = \left(\frac{12t}{B_2^2 + 7B_2 + 12}\right)^{B_2} \left[\frac{B_1^2 + 5B_1 + 6}{12(B_1 - 1)D^{B_2 - B_1}} - \frac{B_2 - B_1}{(B_1 - 1)(B_2 - 1)D^{B_2 - 1}} - \frac{1}{(B_2 - 1)t^{B_2 - 1}} \right]$$

(34)

$$t_0 = \frac{12t}{B_2^2 + 7B_2 + 12} \tag{35}$$

$$C_2 = t_0^{B_2} \tag{36}$$

$$C_1 = n_1 = C_2/D^{B_2 - B_1} \tag{37}$$

$$n_D = C_2/D^{B_2} \tag{38}$$

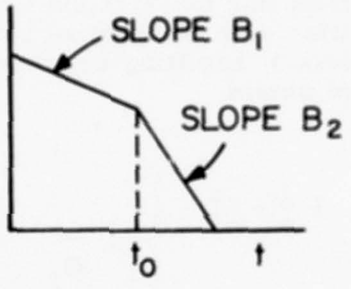

FIG. 6. A TWO SLOPE PIECEWISE - LINEAR GENERALIZATION OF ZIPF'S LAW.

In Appendix A we show that the basic form of Zipf's law, as well as other important probability distributions, can be derived from a linear graph model.

3.0 Zipf's Laws Applied to Computer Languages

3.1 Introduction

In Section 2.1 we began our discussion of the analogies between natural language and computer language. In essence, the analogy is close enough so that we suspect at the outset that computer language will obey one of the original or generalized forms of Zipf's law. Other convincing evidence would be the development of a stochastic model similar to that presented in Appendix A which generates computer programs. Of course in the final analysis, proof is dependent on how well experimental data plotted on log-log paper fits Zipf's laws and by the prediction accuracy of any formulas developed in the analysis.

3.2 Experimental Evidence

We have examined a number of computer programs and always find that some form of Zipf's law seems to fit the data fairly well. The data are listed in Tables 2 to 12 and the Zipf plots are given in Figs. 7 to 17.

Table 2 - Experimental Data on Computer Programs and Zipf's Laws

	Program	Figures		Types	Tokens
1.	PL/I Fibonacci: Numbers Program - 11 Lines long, on p. 81 of Ref. 8	7 8 9	Operators Operands Operators & Operands	12 9 21	31 Operators 24 Operands 55 Total
2.	PL/I Student Grades Program - 27 lines long, on p. 178 of Ref. 8	10 11 12		20 31 51	117 Operators 105 Operands 222 Total
3.	MIKBUG - Executive Program for M6 micro-processor	13		25 62 87	190 Operators 132 Operands 322 Total
4.	PDP-11 Assembly Language Programs	14	Operators	39	1572 Tokens
5.	Variable Names in 3 PL/I Programs	15	Codes	50, 24, 34	370, 238, 193
6.	FORTRAN Statement Types	16		40	200k, and 10k
	(Table 2, Ref. 18)	17			

In program No. 1 we have separately analyzed operators, operands, and combined operators and operands. (See Tables 3, 4, 5 and Figs. 7, 8, 9.) Note that we have followed Halstead[17] in our definition of operators and operands. Basically in PL/I operators include: key words, comparison operators, ";", and ":". The operands include: variables and constants. Excluded from both lists are declare statement and comments. Labels are considered part of the GO TO statement, i.e., GO TO Label 1 is an operator. Labels which are not used are ignored as are comments. Note that a simple Zipf's law fits operators and operands well (cf. Figs. 7, 8); however the combination of operators and operands is better fitted by the first generalized form of Zipf's law (cf. Fig. 9).

In program No. 2 we find that neither the operators (cf. Table 6) nor the operands (cf. Table 7) fit Zipf's law too well (cf. Figs. 10, 11).

Table 3 - Rank-Probability Data for Operators of Program No. 1 in Table 2

r-Rank	p_r-Probability	Symbol	Count
1	0.35	;	11
2	0.16	=	5
3	0.10	,	3
4	0.065	:	2
5	0.065	PUT LIST	2
6	0.065	+	2
7	0.032	PROCEDURE	1
8	0.032	GET LIST	1
9	0.032	IF THEN	1
10	0.032	<	1
11	0.032	GO TO	1
12	0.032	END	1
		Total	31

Table 4 - Rank-Probability Data for Operands of Program No. 1 in Table 2

r-Rank	p_r-Probability	Symbol	Count
1	0.25	LAST-F	6
2	0.17	N	4
3	0.17	PREV F	4
4	0.083	FIBONACCI	2
5	0.083	LIMIT	2
6	0.083	REPEAT	2
7	0.083	TEMP	2
8	0.042	2	1
9	0.042	1	1
		Total	24

6.4 Language Theory Measures of Complexity

The use of number of statements as a measure of complexity is often used. It is recognized that such a simplistic measure does not truly gauge the program. Often there are programs with a few lines considerably more complex than programs with many lines. As a better measure Halstead (Reference 27) suggested operator and operand count. The analogy between operators, operands and verb nouns suggested the application of Zipf's Law from natural languages to programming languages.

The relationships between number of operators and operand types, as well as their frequency of occurrence, was shown to follow the basic Zipf's Law. An extended form of Zipf's Law has been derived which more closely models some of the experimental data.

The theory culminates in an equation which relates operator and operand length to the number of types (References 9 and 25). If we estimate early in the design the number of input and output varibles and operators (which will be used in the program) we can obtain an estimate of the program length.

These results correlate well with the results which Halstead has obtained using software science techniques (Reference 21, Chapter 3, and Reference 27).

Table 6 - Rank-Probability Data for Operators of Program No. 2 in Table 2

r-Rank	p_r-Probability	Symbol	Count
1	0.31	=	36
2	0.23	;	28
3	0.14	IF THEN	17
4	0.043	,	5
5	0.034	SUBSTR	4
6	0.026	:	3
7	0.026	( )	3
8	0.026	SKIP	3
9	0.026	+	3
10	0.026	PUT LIST	3
11	0.017	| |	2
12	0.017	GO TO	2
13	0.0085	PROCEDURE OPTIONS MAIN	1
14	0.0085	PAGE	1
15	0.0085	LINE	1
16	0.0085	GET LIST	1
17	0.0085	*	1
18	0.0085	/	1
19	0.0085	-	1
20	0.0085	END	1
		Total	117

Table 7 - Rank-Probability Data for Operands of Program No. 2 in Table 2

r-Rank	p_r-Probability	Symbol	Count
1	0.086	1	9
2	0.057	FINAL-LETTER-GRADE	6
3	0.057	STUDENT DATA	6
4	0.057	MID-TERM	6
5	0.057	FINAL	6
6	0.057	MIDTERM-TEST	6
7	0.057	FINAL-TEST	6
8	0.057	FINAL-GRADE	6
9	0.038	POSITION	4
10	0.038	3	4
11	0.038	2	4
12	0.038	0	4
13	0.029	B+	3
14	0.029	A	3
15	0.029	4	3
16	0.029	B	3
17	0.029	C	3
18	0.029	D	3
19	0.029	F	3
20	0.019	FINAL-GRADES	2
21	0.019		2
22	0.019	STUDENT-NAME	2
23	0.019	PROCESS-NEXT-STUDENT	2
24	0.019	TERMINATE	2
25	0.0095	'THE FINAL...COURSE'	1
26	0.0095	10	1
27	0.0095	8	1
28	0.0095	44	1
29	0.0095	-	1
30	0.0095	0.5	1
31	0.0095	INDEX	1
		Total	105

Table 8 - Rank-Probability Data for Operators Plus Operands of Program No. 2 in Table 2

r-Rank	p_r-Probability	Symbol	Count
1	0.16	=	36
2	0.13		28
3	0.077	IF THEN	17
4	0.041	1	9
5	0.027	FINAL-LETTER-GRADE	6
6	0.027	STUDENT-DATE	6
7	0.027	MID-TERM	6
8	0.027	FINAL	6
9	0.027	MIDTERM-TEST	6
10	0.027	FINAL-TEST	6
11	0.027	FINAL-GRADE	6
12	0.023		5
13	0.018	SUBSTR	4
14	0.018	POSITION	4
15	0.018	3	4
16	0.018	2	4
17	0.018	0	4
18	0.014	:	3
19	0.014	( )	3
20	0.014	SKIP	3
21	0.014	+	3
22	0.014	PUT LIST	3
23	0.014	b	3
24	0.014	A	3
25	0.014	4	3
26	0.014	B	3
27	0.014	C	3
28	0.014	D	3
29	0.014	F	3
30	0.009	| |	2
31	0.009	GO TO	2
32	0.009	FINAL-GRADES	2
33	0.009	'	2
34	0.009	STUDENT-NAME	2
35	0.009	PROCESS-NEXT-STRING	2
36	0.009	TERMINATE	2
37	0.0045	PROCEDURE OPTIONS MAIN	1
38	0.0045	PAGE	1
39	0.0045	LINE	1
40	0.0045	GET LIST	1
41	0.0045	*	1
42	0.0045	/	1
43	0.0045	-	1
44	0.0045	END	1
45	0.0045	'THE FINAL...GRADE'	
46	0.0045	10	1
47	0.0045	8	1
48	0.0045	44	1
49	0.0045	-	1
50	0.0045	0.5	1
51	0.0045	INDEX	1
		Total	221

Table 9 - Listing of Fibonacci Numbers PL/I Program
(p. 81 of Ref. 8)

```
FIBØNACCI:  PRØCEDURE;
            N = 2;
            GET LIST (PREV_F, LAST_F, LIMIT);
            PUT LIST (PREV_F, LAST_F);
  REPEAT:   TEMP = LAST_F;
            LAST_F = LAST_F + PREV_F;
            PREV_F = TEMP;
            PUT LIST (LAST_F);
            N = N+1;
            IF N < LIMIT THEN GØ TØ REPEAT;
            END FIBØNACCI;
```

```
FINAL-GRADES: PROCEDURE OPTIONS (MAIN);
  /* THE PROCEDURE CALCULATES THE FINAL GRADES THAT WILL BE GIVEN IN A
     PL/1 COURSE. USING THE LETTER GRADES RECEIVED IN THE MIDTERM TEST
     AND THE FINAL TEST OF THE COURSE, THE FINAL GRADE IS CALCULATED
     FOR EACH STUDENT. */

  /* THE NAME AND THE GRADES OF A STUDENT ARE KEPT IN A CHARACTER
     STRING VARIABLE. */
  DECLARE STUDENT-DATA CHAR (30) VARYING;
  DECLARE POSITION FIXED DECIMAL(2,0); /* INDEX INTO THE DATA. */

  /* THE LETTER GRADES ARE PLACED IN CHARACTER STRING VARIABLES OF
     LENGTH ONE. NUMERIC GRADES ARE KEPT IN INTEGER VARIABLES. */
  DECLARE (MIDTERM, FINAL, FINAL-LETTER-GRADE) CHAR (1);
  DECLARE (MIDTERM-TEST, FINAL-TEST, FINAL-GRADE) FIXED DEC (2);

  /* A CHARACTER STRING VARIABLE OF FIXED LENGTH IS USED FOR THE
     PRINTING OF THE STUDENT NAME. IT WILL PROVIDE ALIGNMENT IN THE
     OUTPUT STREAM. */
  DECLARE STUDENT-NAME CHARACTER (15);

  /* PREPARE THE PAGE HEADING. */
  PUT LIST ('THE FINAL GRADES RECEIVED IN THE PL/1 COURSE')PAGE LINE(10);
  PUT LIST (19)' ' || (44)'-') SKIP (0);
  PUT SKIP (1);

PROCESS-NEXT-STUDENT:
  /* ACQUIRE THE DATA OF ONE STUDENT. */
  GET LIST (STUDENT-DATA);

  /* THE END OF THE INPUT STREAM IS REACHED WHEN A NULL STRING IS
     ENCOUNTERED DURING THE DATA ACQUISITION. */
  IF STUDENT-DATA = '' THEN GO TO TERMINATE;

  /* LOCATE THE END OF THE STUDENT NAME. */
  POSITION = INDEX (STUDENT-DATA, ',');

  /* LOCATE THE MIDTERM AND THE FINAL TEST RESULTS. */
  MIDTERM = SUBSTR (STUDENT-DATA, POSITION + 1, 1);
  FINAL = SUBSTR (STUDENT-DATA, POSITION + 3, 1);

  /* CONVERT THE MIDTERM AND THE FINAL TEST RESULTS INTO NUMERIC
     GRADES. */
  IF MIDTERM = 'A' THEN MIDTERM-TEST = 4;
  IF MIDTERM = 'B' THEN MIDTERM-TEST = 3;
  IF MIDTERM = 'C' THEN MIDTERM-TEST = 2;
  IF MIDTERM = 'D' THEN MIDTERM-TEST = 1;
  IF MIDTERM = 'F' THEN MIDTERM-TEST = 0;
  IF FINAL = 'A' THEN FINAL-TEST = 4;
  IF FINAL = 'B' THEN FINAL-TEST = 3;
  IF FINAL = 'C' THEN FINAL-TEST = 2;
  IF FINAL = 'D' THEN FINAL-TEST = 1;
  IF FINAL = 'F' THEN FINAL-TEST = 0;

  /* CALCULATE THE FINAL GRADE THE STUDENT WILL RECEIVE. */
  FINAL-GRADE = (MIDTERM-TEST + 2 * FINAL-TEST) /3 + 0.5;

  /* DETERMINE THE FINAL LETTER GRADE. */
  IF FINAL-GRADE = 4 THEN FINAL-LETTER-GRADE = 'A';
  IF FINAL-GRADE = 3 THEN FINAL-LETTER-GRADE = 'B';
  IF FINAL-GRADE = 2 THEN FINAL-LETTER-GRADE = 'C';
  IF FINAL-GRADE = 1 THEN FINAL-LETTER-GRADE = 'D';
  IF FINAL-GRADE = 0 THEN FINAL-LETTER-GRADE = 'F';

  /* DISPLAY THE FINAL GRADE OF THE STUDENT. */
  STUDENT-NAME = SUBSTR (STUDENT-DATA, 1, POSITION -1);
  PUT LIST ('', STUDENT-NAME || FINAL-LETTER-GRADE) SKIP;

  GO TO PROCESS-NEXT-STUDENT; /* GET THE DATA OF THE NEXT STUDENT. */
TERMINATE: END FINAL-GRADES;
```

Table 10 - Listing of Students Grades PL/I Program
(p. 178 of Ref. 8)

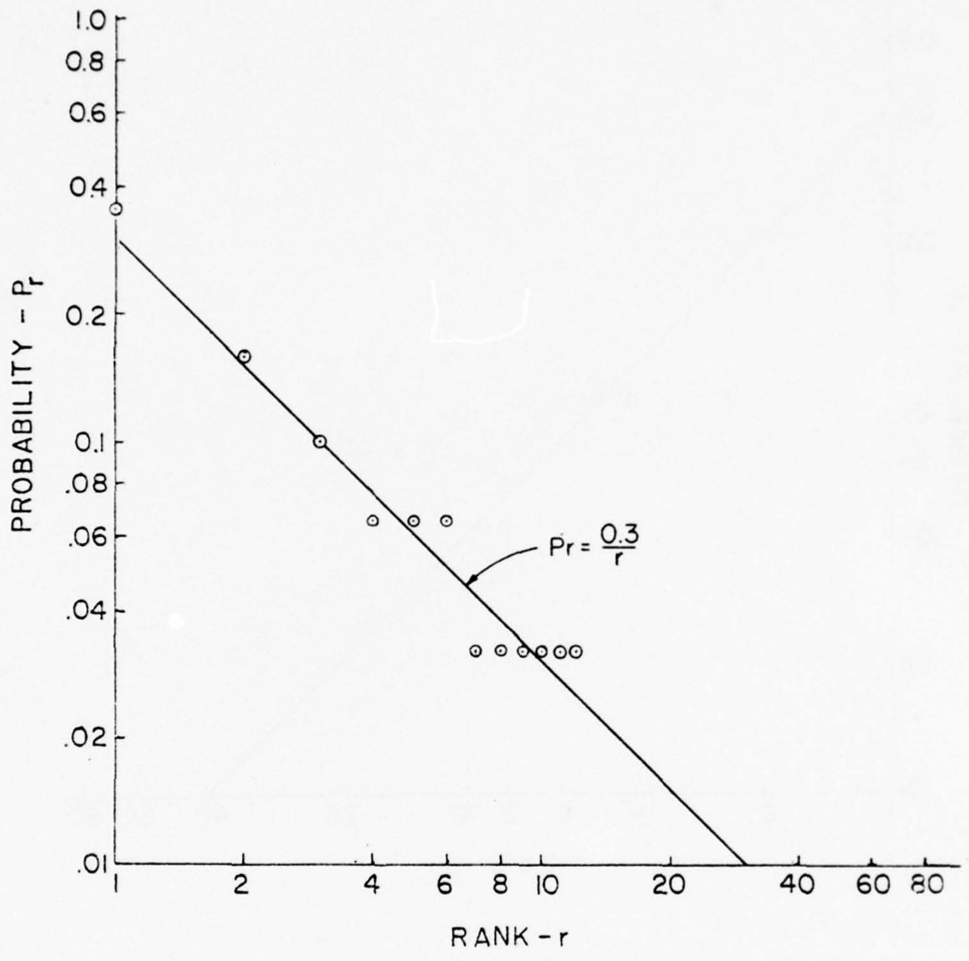

FIG. 7. ZIPF LAW PLOT FOR OPERATORS IN PROGRAM NO. 1 OF TABLE 2.

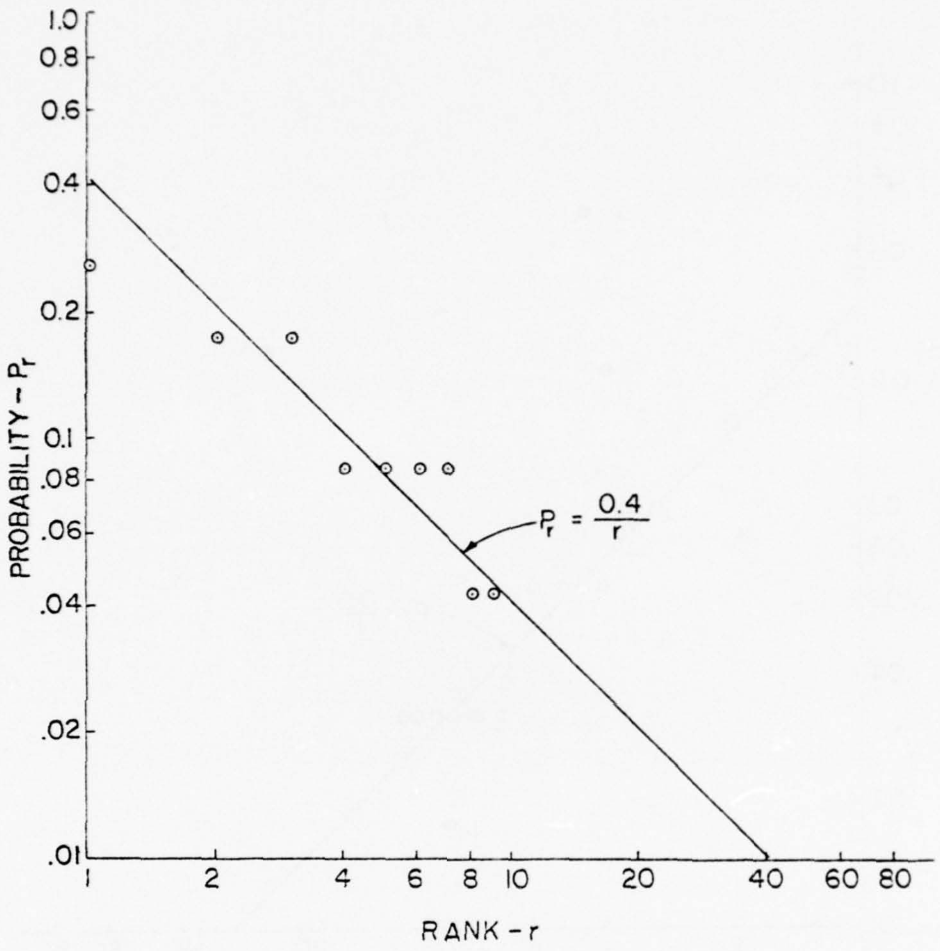

FIG. 8. ZIPF LAW PLOT FOR OPERANDS IN PROGRAM NO. 1 OF TABLE 2.

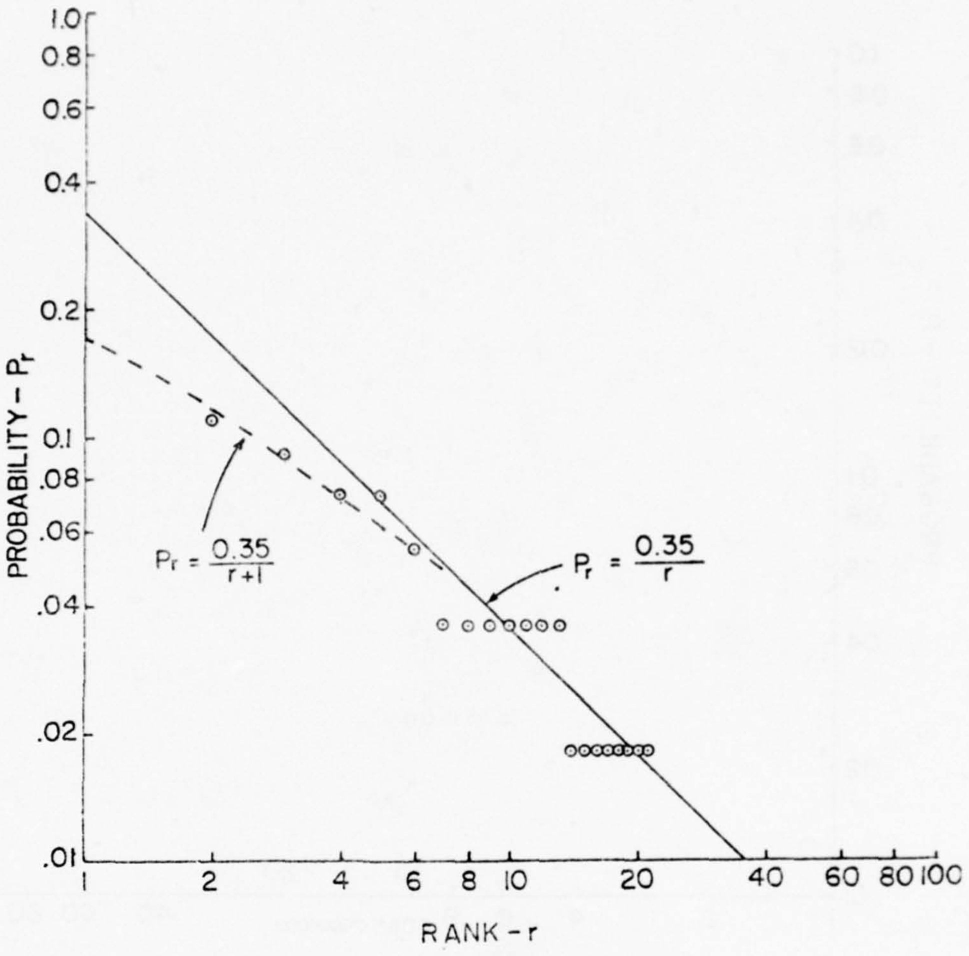

FIG. 9. ZIPF LAW PLOT FOR COMBINED OPERATORS AND OPERANDS IN PROGRAM NO. 1 OF TABLE 2.

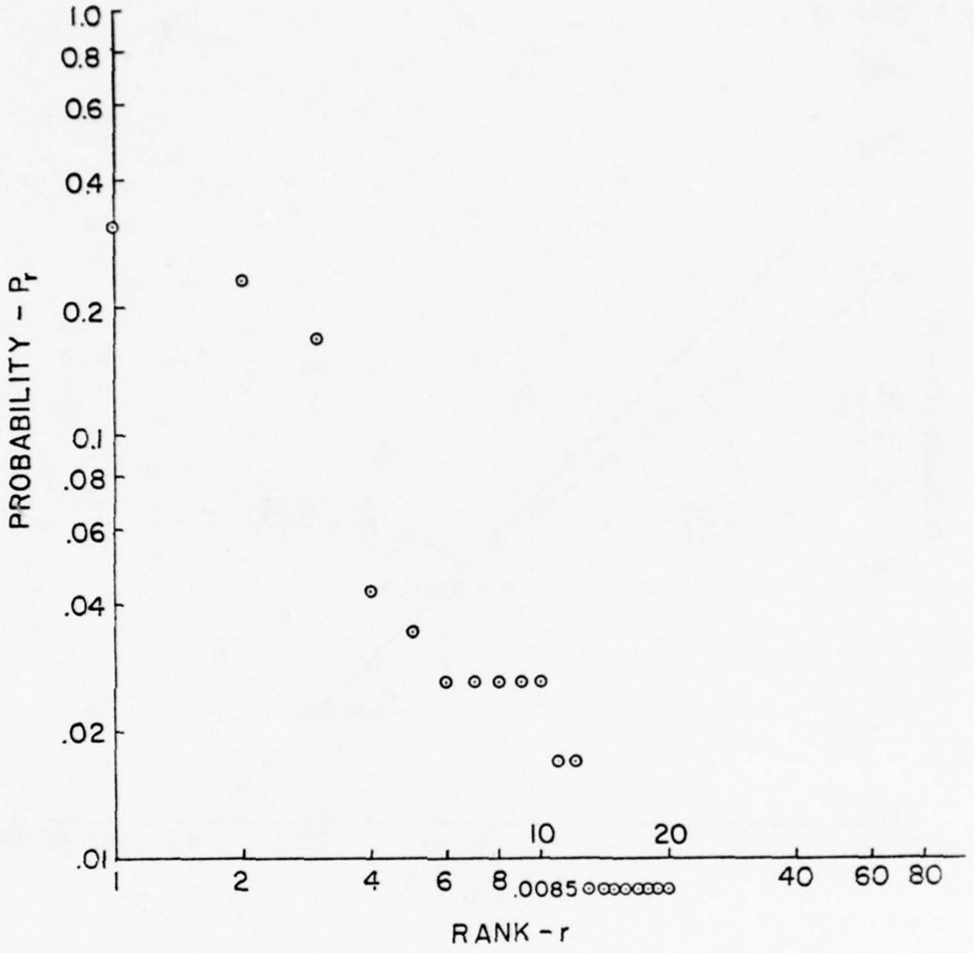

FIG. 10. ZIPF LAW PLOT FOR OPERATORS IN PROGRAM NO. 2 OF TABLE 2.

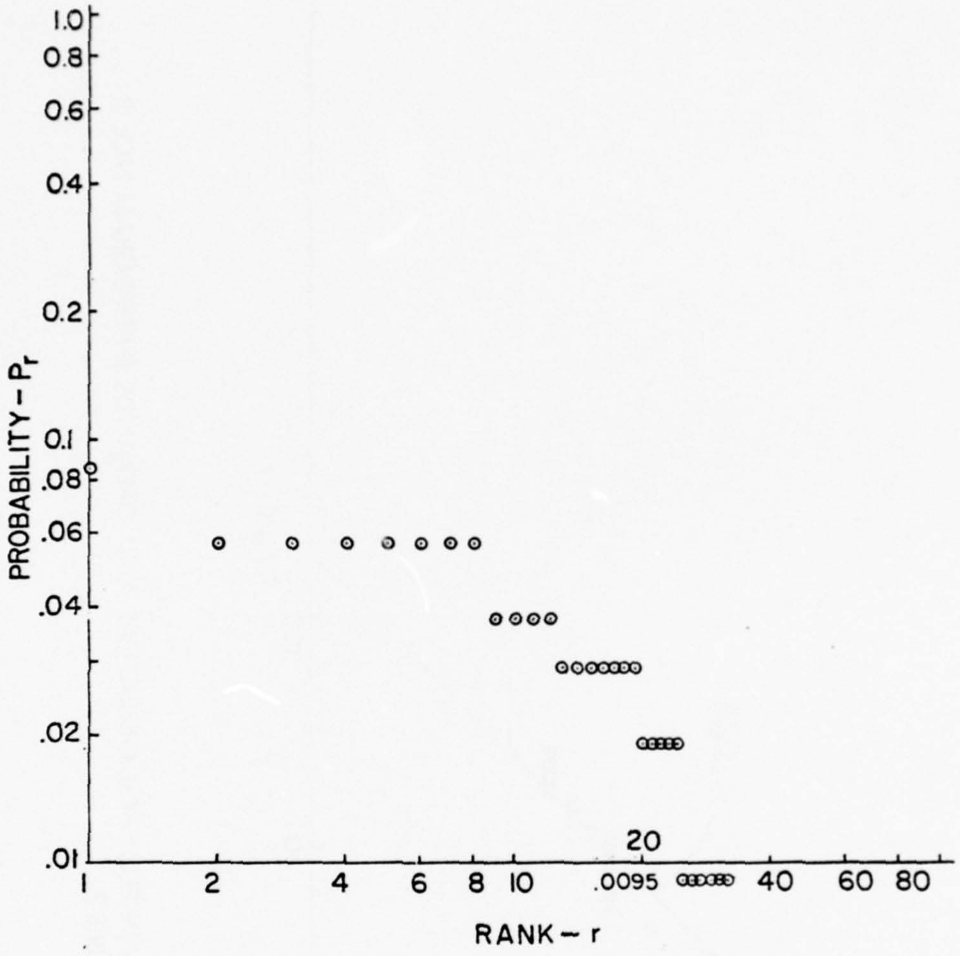

FIG. 11. ZIPF LAW PLOT FOR OPERANDS IN PROGRAM NO. 2 OF TABLE 2.

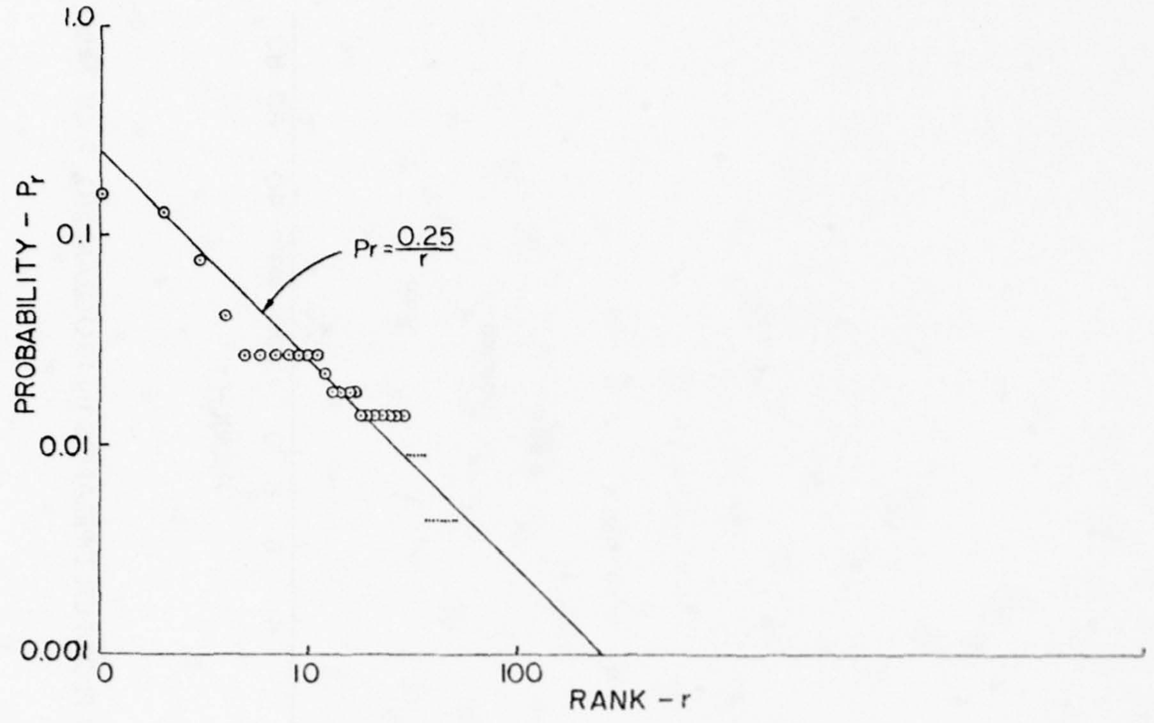

FIG. 12. ZIPF LAW PLOT FOR OPERATORS AND OPERANDS IN PROGRAM NO. 2 OF TABLE 2.

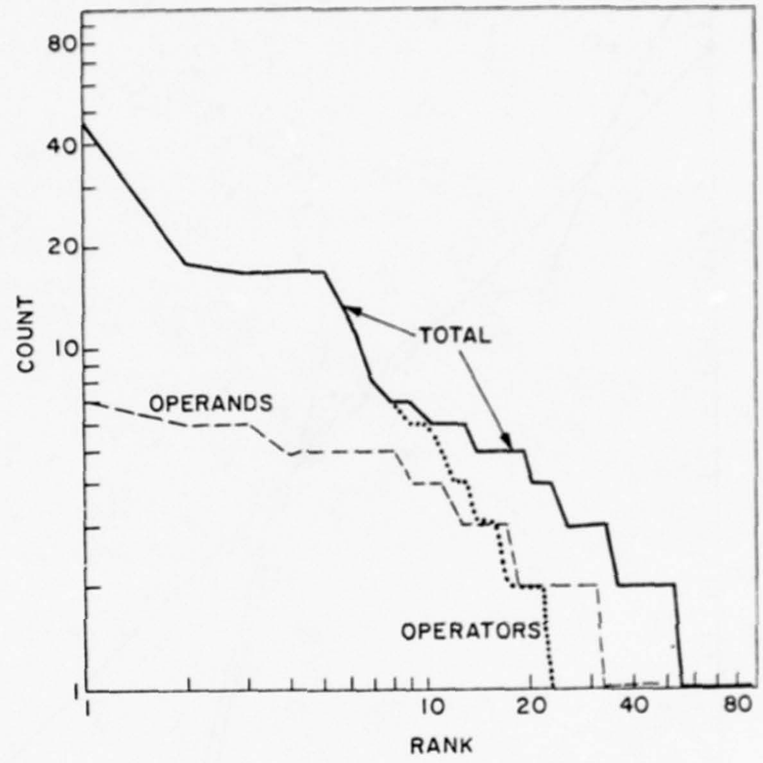

FIG. 13. RELATION BETWEEN THE RANK AND COUNTS OF OPERATORS, OPERANDS, AND THEIR TOTAL.

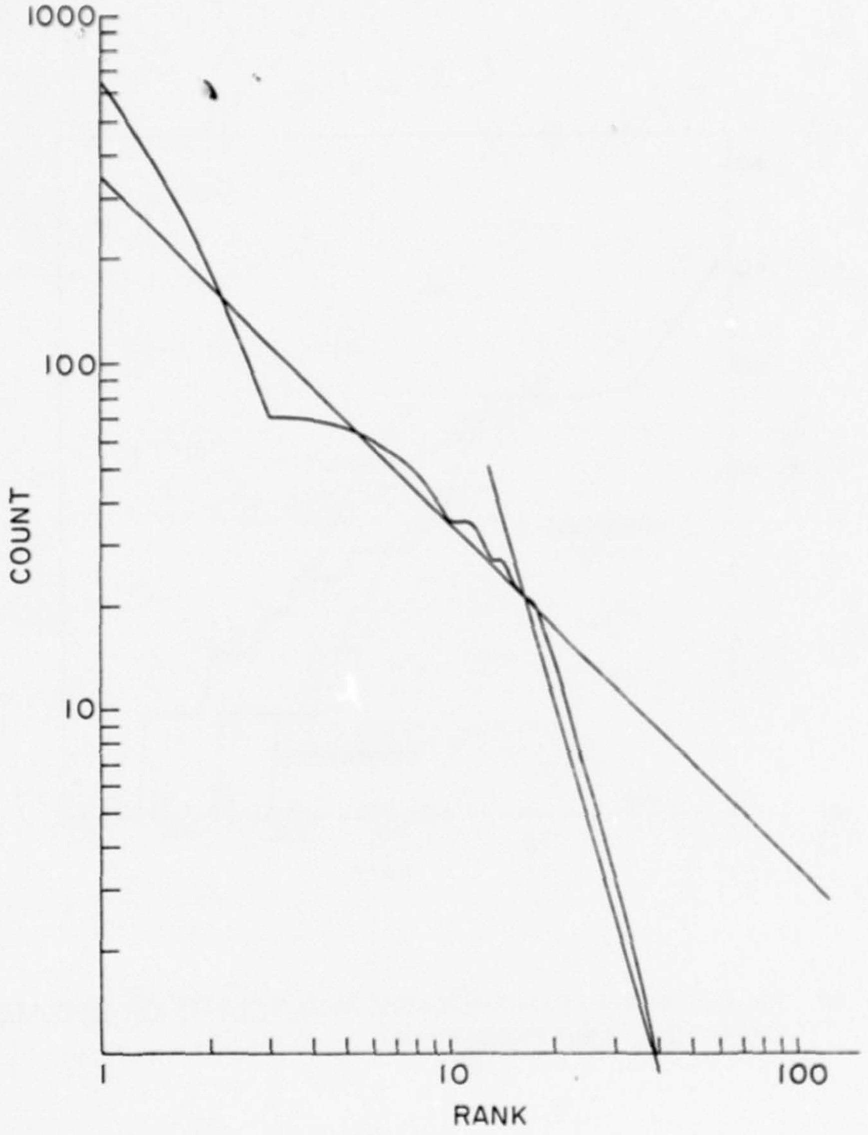

FIG. 14. RELATION BETWEEN THE RANK AND OP CODE COUNTS FOR SEVERAL PDP-11 PROGRAMS.

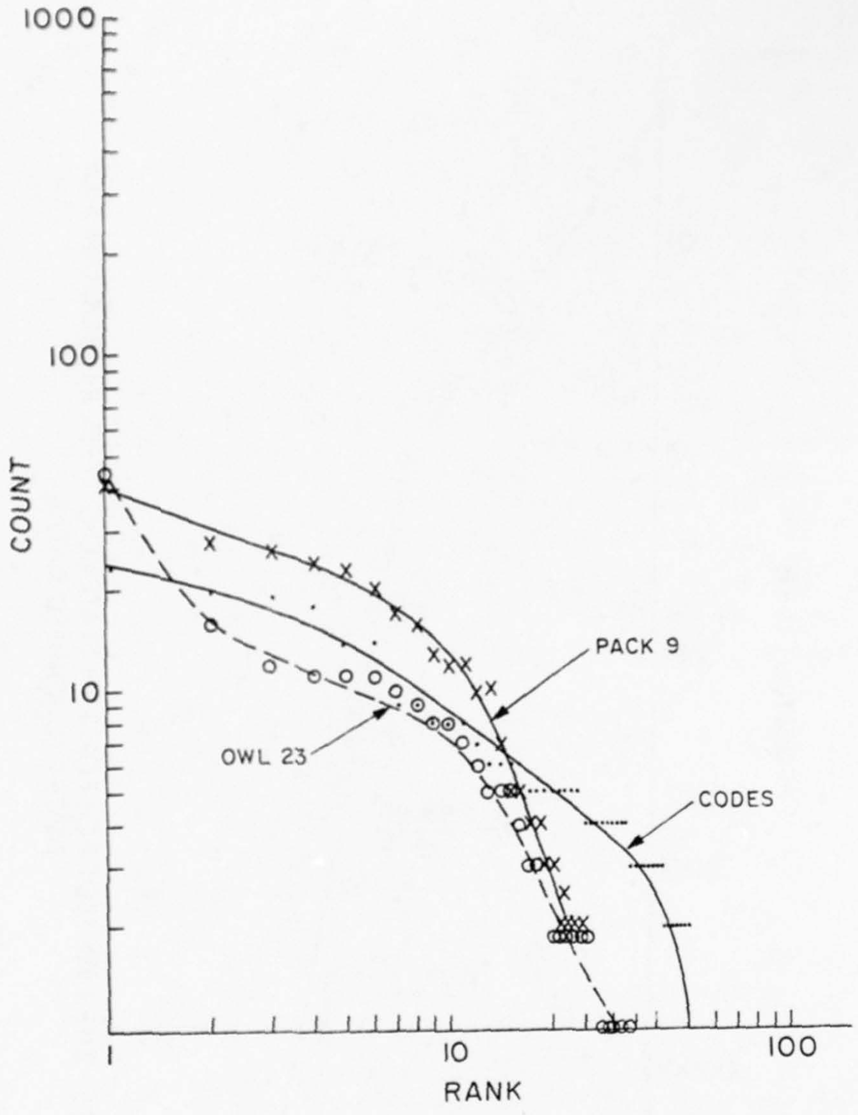

FIG. 15. PLOT OF THREE PL/I PROGRAMS.

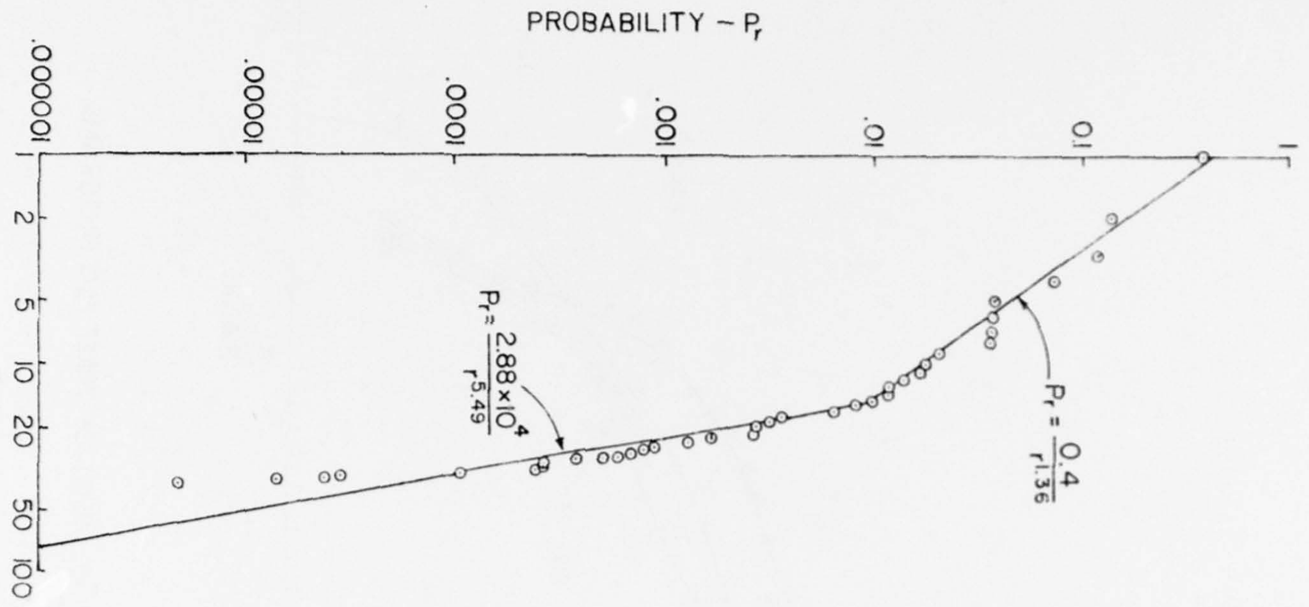

FIG. 16. ZIPF LAW PLOT FOR FORTRAN STATEMENT TYPES IN PROGRAM 5 OF TABLE 2 (LOCKHEED DATA).

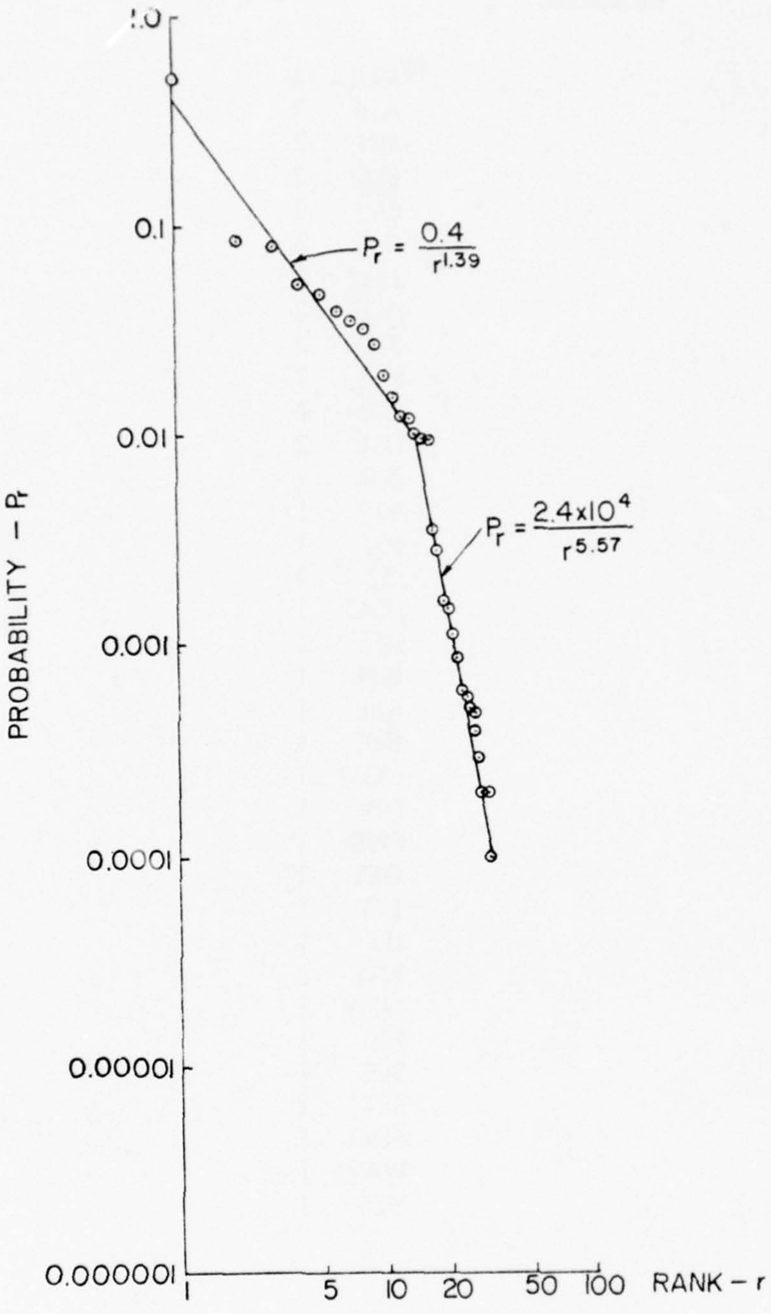

FIG. 17. ZIPF LAW PLOT FOR FORTRAN STATEMENT TYPES IN PROGRAM 5 OF TABLE 2 (STANFORD DATA).

Table 11 - Operator Counts for Three Letter OP Codes Plotted in Fig. 14. (Also includes system calls, macros, assembler directives, etc. which were not included.

MOV	644	MUL	4
JSR	186	ASR	3
GEN	98	BHI	3
JMP	70	BLO	3
CLR	68	BMI	3
RTS	64	GPL	3
ADD	56	20S	2
BEQ	51	;WE	2
CMP	47	NOT	2
ASL	39	ROL	2
SYS	39	RUN	2
BNE	33	USE	2
SOB	33	0.0	1
BIT	31	40S	1
BCS	27	80.	1
DEC	27	;32	1
BCC	22	;GO	1
TST	21	;IF	1
BIC	20	GPL	1
SUB	20	ALL	1
INC	17	BVC	1
BGE	12	COM	1
BIS	12	DIV	1
THE	12	FWD	1
FOR	11	GET	1
ARG	9	I;O	1
BLT	9	ITS	1
END	8	N-1	1
AND	5	NEW	1
ASH	5	PTR	1
BGT	5	SEE	1
BLE	5	SET	1
NEG	5	SUM	1
ROR	5	WAY	1
BPL	4	YES	1

Table 12 - Rank-Frequency Data for FORTRAN Statement Types for Program 5 of Table 2

	← Lockheed Data →			← Stanford Data →		
Statement Type	Rank	Probability*	Number	Rank	Probability*	Number
Assignment	1	0.38	78435	1	0.50	4869
IF	2	0.14	27967	2	0.084	816
GO TO	3	0.120	24942	3	0.080	777
Call	4	0.073	15125	7	0.035	339
Continue	5	0.038	9165	8	0.032	309
Write	6	0.037	7795	4	0.052	508
Format	7	0.037	7685	6	0.039	380
DO	8	0.636	7476	5	0.047	457
Data	9	0.021	4468	18	0.0029	28
Return	10	0.018	3639	10	0.019	186
Dimension	11	0.017	3492	11	0.015	141
Common	12	0.014	2908	9	0.027	263
End	13	0.012	2565	12	0.012	121
Buffer	14	0.012	2501	-	-	0
Subroutine	15	0.010	2001	15	0.0095	93
Rewind	16	0.0083	1724	23	0.00062	6
Equivalence	17	0.0066	1382	13	0.0012	113
Endfile	18	0.0037	765	29	0.0002	2
Integer	19	0.0032	657	17	0.0035	34
Read	20	0.0028	586	16	0.0094	92
Encode	21	0.0028	583	-	-	0
Decode	22	0.0027	557	-	-	0
Print	23	0.0017	345	25	0.00052	5
Entry	24	0.0013	279	20	0.0015	15
Stop	25	0.0091	190	21	0.0011	11
Logical	26	0.0082	170	22	0.0009	9
Real	27	0.00071	147	28	0.0003	3
Ident	28	0.00051	106	-	-	0
Overlay	29	0.00039	82	-	-	0
Pause	30	0.00027	57	24	0.0006	6
Assign	31	0.00027	57	27	0.0004	4
Punch	32	0.00025	52	26	0.0005	5
External	33	0.00011	23	31	0.0001	1
Complex	34	0.000029	6	-	-	0
Name List	35	0.000024	5	-	-	0
Double	36	0.000014	3	14	0.010	99
Block Data	37	0.0000048	1	30	0.0002	2
Implicit	-	-	0	19	0.0016	16
Input	-	-	0	-	-	0
Output	-	-	0	-	-	0
TOTAL			207,941			9710
Comment		(28)	52,924	(11)		1090
Continuation		(7)		(7)		636

* Knuth counted IF statements twice, thus the total is actually higher than the number of statements.

However, a reasonable fit is obtained for their sum (cf. Table 8 and Fig. 12). The listings of programs No. 1 and 2 of Table 2 appear in Tables 9 and 10.

The MIKBUG is a TTY handler and simple debugger for the M6800 microprocessor. Fig. 13 shows the relations between the rank and the counts of operators (OPcodes), operands (variable names and statement labels) and their total. Fig. 14 shows the relation between the rank and the OP code counts for several PDP-11 programs. Both of these examples are in assembly language. As in most higher languages, the upper part of the operand plot is flattened, and the lower part of the operator plot drops off more rapidly than the upper part.

Three PL/I programs are plotted in Fig. 15. Only variable names are shown. The flattened upper part can again be seen. In one respect computer operands differ quite markedly from their natural language analog nouns, or from words in general. The second form, or low frequency form, of Zipf's law is not followed that well in the computer programs.

We have explored how well PL/I and assembler operators and operands are fitted by Zipf's laws and their extensions. One can also explore how well statement types in FORTRAN fit Zipf's law. Knuth (cf. Ref. 18) studies data on about 200,000 FORTRAN statements collected at Lockheed Aircraft and 10,000 at Stanford. His data is converted to Rank-frequency data in Table 12 and is plotted in Fig. 16 and 17. Note that in Table 12 the statement types rank differently for the Lockheed and Stanford data.

(1) The top 10 ranks are quite similar except for "DATA".

(2) Similarly the top 20 are the same except for three exceptions (14, 23, 29).

(3) Stanford Programmers are either unaware or choose not to use ENCODE, DECODE, IDENT, and OVERLAY.

(4) Stanford Programmers use DOUBLE on occassion, while Lockheed Programmers almost never use it.

(5) Lockheed Programmers produce about 2-1/2 times as many comments as Stanford ones.

(6) The large 0.5 probability of Stanford's assignment statements seems to indicate that the Stanford code is more simplistic (for better or worse?) than the Lockheed code.

The most amazing fact is that with all these differences, the data is reasonably well fitted by a two segment piecewise linear Zipf law over many decades, and the constants C and B are almost identical for both plots.

Based on the above data and our previously drawn analogies between computer and natural languages it is fair to conclude that computer programs do follow Zipf's law and its extensions. Conclusions on how closely the fit is in general, and whether there are any major exceptions or classes of data which fit particular forms of the model, must await further study of data. The use to which we put our Zipf's law models is discussed in the following section.

We can explore how well the formulas for the constant C and the number of tokens n estimate the actual values observed in the data. In Table 13, a comparison is made for 4 cases between the calculated value of $p_1 = C$, and the experimental value $p_1 = n_1/n$, and the model abcissa intercept. Similarly we compare the actual value of n with the calculated estimates. In all cases the agreement is reasonable. Additional data comparing calculated and actual values of n is given in Sec. 3.4.

Table 13

Camparison of Calculated vs. Experimental Values of p_1 and n

		Calculated				
Fig. No.	t	$\frac{1}{0.5772+\ln t}$	$p_1 = \frac{n_1}{n}$	Model Intercept	Calculated $t(0.5772+\ln t)$	Data n
7	12	0.32	0.35	0.3	37.5	31
8	9	0.36	0.25	0.4	25	24
9	21	0.28	0.20	0.35	75	55
12	51	0.22	0.16	0.25	23	222

3.3 Laws of Complexity and Length

The fact that Zipf's law can be applied to computer programs is of interest but of little practical importance by itself. Also being able to calculate program length from the number of types appearing in the program seems unimportant unless the relation used forms a part of a more comprehensive theory of program complexity.* Such a theory is being developed,[19] and a sketch will be given here. It is hoped that by this means the number of programming errors, development effort, etc., can be related to other parameters.

Briefly, a computer program indicates to the computer which function to use in mapping input data to output data. The functions of practical interest are defined in terms of simpler functions, and these in terms of yet simpler functions, etc., until functions are reached which are evaluated by

* Of course there are many obvious uses of such a length measure which suggest themselves even without such a general theory: (1) to use as an improved normalizing factor instead of statement length in comparing debugging data[21], (2) as a managerial tool in comparing and ranking programs by complexity, (3) as a parameter to use in correlation studies of experimental data on number of bugs, man hours, etc.

the computer hardware or by system subroutines. The complexity of a function can be measured by the extent to which it can be decomposed into simpler functions, as is well known in switching theory and recursive function theory. The number of levels in the functional decomposition will be related to the number of variable and procedure names used in the program. Some interesting preliminary results have been obtained on the tradeoffs between program length, number of input and output bits, hardware complexity, storage requirements, and execution time.[19] For example: 1) the product of execution time and hardware complexity is approximately constant; 2) a minimum expected program length can be defined, but it depends on the probability of using the function, not its complexity; 3) trying to reach the minimum program length might require excessively large "compiler" complexity and storage space. Some definite bounds relating the above parameters have been obtained and will be reported elsewhere (Ref. 19). Of course, the quantitative results apply strictly only to the stated mathematical model and their application to actual programming situations will have to be studied.

One method of initially estimating program length* (number of tokens) is to estimate the number of types. We assume the analyst initially has a complete description of the problem and that a partial analysis and choice of key algorithms has been made. An elementary approach might be to estimate the token size by

(1) Estimating the number of operator types which will be used in the language by the assigned programmers.

(2) Estimate the number of input variables, output variables, intermediate variables, and constants which will be needed.

(3) Sum the estimates of step (1) and (2) and substitute in Eqs. 16, 20, 26, or 34.

3.4 Relationship to "Software Physics"

The initial motivation for the application of Zipf's law to computer languages came from a review of Halstead's[17] work on Software Physics. Early in his work he arrives at a formula for program length

$$L = \eta_1 \log_2 \eta_1 + \eta_2 \log_2 \eta_2 \qquad (39)$$

where

$L \equiv$ Program length

$\eta_1 \equiv$ Number of operator types

$\eta_2 \equiv$ Number of operand types

---

* In addition one must add other classes of statements and programming elements such as: comments, declares, certain assembler directives, etc.

In terms of our notation the analogous quantities are

$$t = \eta_1 + \eta_2 \tag{40}$$

$$\dot{n} = L \tag{41}$$

Note that Eq. 39 and Eqs. 14 and 20 are of similar form. In Table 14, we compare the actual number of tokens with the number of tokens calculated using Eqs. 14, 20, and 39. Both the average error and average magnitude error are computed. Both Eqs. 14 and 39 yield good agreement between actual and calculated results.

Conclusion

A connection has been established between some well-known results in the statistics of natural languages (Zipf's law) and some recent observations on the relations between various parameters of computer programs, mostly initiated by the work of Halstead. Also, a mathematical model is presented which can yield both the original Zipf law and the binomial distribution as special cases. This derivation is a generalization of that of Mandelbrot. The model can be applied to such diverse processes as the generation of a computer program by a contex free grammar, and to the summation of random variables (central limit theorem). Work is continuing on relating the length messages obtained to the number of programming bugs and man-hours of effort.

Table 14 - Comparison of Length Estimation Formulas

Sample Source & Type	Actual Number of Tokens n	Estimate 3		Estimate 1		Estimate 2	
		Eq. 39	% Error	Eq. 14	% Error	Eq. 20	%Error
Lawson-11 Statements PL/I	55	72	+31	76	+38	46	-16
Lawson-27 Statements PL/I	222	240	+8	230	+3.6	140	-37
MIKBUG-M6800 Micro Exec. Program	87	481	+49	378	+17	230	-29
Halstead Ex. Ref. (17), p. 10	52	49	-6	32	-38	20	-62
Halstead, Ref. (17) CACM #1	104	104	0	109	+5	66	-36
CACM #2	82	77	-6	89	+8	54	-34
CACM #3	453	300	-34	275	-39	167	-63
CACM #4	132	139	+5	155	+17	94	-28
CACM #5	123	123	0	124	+1	75	-39
CACM #6	98	101	+3	109	+12	61	-37
CACM #7	59	62	+5	67	+13	40	-31
CACM #8	131	171	-11	124	-5	75	-43
CACM #9	314	288	-8	280	-11	170	-46
CACM #10	46	52	+13	62	+35	38	-17
CACM #11	53	52	-1	62	+17	38	-28
CACM #12	59	62	+5	71	+20	43	-27
CACM #13	59	57	-3	71	+20	43	-17
CACM #14	186	163	-12	170	-9	103	-45
Average Error	-	-	+2.1%	-	+5.8%	-	-33%
Average Magnitude Error	-	-	10%	-	17.2%	-	33%

Appendix A - Derivation of Zipf's and Other Related Laws

A. 1 Origins of Zipf's Law

By Zipf's law we mean a rank-probability distribution such as $p_r = c/r$, or a generalization such as $p_r = C/(A+r)^B$, or a segmented straight line function on log-log paper. This law is usually attributed to Zipf in connection with word frequencies; however, others have priority not only in linguistics but also in other areas such as incomes, particle size, population, etc. An analogy may be drawn to the central limit theorem, in that many physical processes in several fields imply a simple common probability distribution. The advantages in using such a distribution, if it is shown to be valid in the area under investigation, is that such properties as information content (entropy) or the type-token ratio may be calculated from 2 to 3 easily estimated parameters instead of from perhaps thousands of individual probabilities. A model will be introduced which yields both Zipf's law and the binomial distribution as marginal cases. The model is a generalization of that suggested by Mandelbrot and others.

A. 2 Linear Graph Model

A linear graph model will be described which can give rise to any probability distribution over a countable number of events, then certain particular cases will be identified with some familiar probability distributions, and finally several physical interpretations of the linear graph will be given. Consider a directed linear graph with no circuits (paths returning to their starting node) and with but one origin (node with only leaving branches). Divide the nodes into 3 classes: an origin node which has only branches leaving it, terminal nodes which have no branches leaving them, and intermediate nodes which have one or more branches entering from other nodes and leaving for other nodes. Each intermediate node j will have associated with it a probability distribution $p_i(j)$ giving the probability of leaving by one of the n_j branches:

$$\sum_{i=1}^{n_j} p_i(j) = 1 \quad . \tag{A. 1}$$

Thus each path from the origin passing thru nodes 1 (the origin), $j_2, j_3, \ldots, j_m$ has a probability

$$p_{i_1}(1)\, p_{i_2}(j_2) \cdots p_{i_{m-1}}(j_{m-1}) \tag{A-2}$$

provided that the branch decision probabilities at each node are independent. Finally, each node will have associated with it a probability p_j which is the sum of all of the path probabilities from the origin to it. The sum of the p_j over the terminal nodes (index set T) is clearly unity:

$$\sum_{j \in T} p_j = 1 \quad . \tag{A.3}$$

It is the distribution p_j which will be of interest here.

It can easily be shown that the branch probabilities $p_i(j)$ can be chosen so as to yield any desired probability distribution p_j. This will usually require the $p_i(j)$ to be different for each node j. If there are only a few different distributions $p_i(j)$ allowed, then the p_j over the terminal nodes will belong to one of a few regular distributions.

A. 3 Generating Graph is a Tree

If the generating graph has no reentrant paths, i. e., if there is a unique path from the origin to a given node, then the resulting probability distribution p_j is easier to characterize, and it will be a Zipf type if the resulting tree is homogeneous (branch probabilities independent of node).

The probabilities p_j are easy to calculate in rank order in this case i. e., so that

$$p_1 \geq p_2 \geq p_3 \geq \cdots \quad . \tag{A.4}$$

This calculation is facilitated by associating with each branch a length $\ln \frac{1}{p_i(j)} = \ell_i(j)$. Then the log probability associated with a node is simply the (unique) distance from the origin. If the tree is homogeneous, let n be the number of branches leaving each intermediate node and going to another intermediate node, and let ℓ_i be their lengths with

$$0 < \ell_1 < \ell_2 < \ldots < \ell_n \; .$$

The number of intermediate nodes at a distance of x or less from the origin, R(x), will then satisfy the following difference equation

$$\left.\begin{aligned} R(x) &= \sum_{i=1}^{n} R(x - \ell_i) + 1 & \qquad x &\geq 0 \\ R(x) &= 0 & \qquad x &< 0 \end{aligned}\right\} \tag{A.5}$$

This can be proved by noting that every intermediate node (except the origin) within radius x is contributed by a branch of some length ℓ_i and by no other. The above difference equation has a solution which is asymptotic to

$$\left.\begin{aligned} &R_0(x) = e^{\frac{x}{B}} \\ &\text{where } B \text{ is given by } \sum_{i=1}^{n} e^{-\frac{\ell_i}{B}} = 1 \end{aligned}\right\} \qquad \text{(A. 6)}$$

and in fact $R_0(x)$ is a good smoothed approximation to the discontinuous step function $R(x)$ for all x.

Now let there be t branches going to terminal nodes from each intermediate node (including the origin), and let their probabilities be p_{n+1}, $p_{n+2}, \ldots, p_{n+t}$. We must have

$$\sum_{i=1}^{n} p_i + \sum_{i=n+1}^{n+t} p_i = 1 \qquad \text{(A. 7)}$$

$$\sum_{i=1}^{n} e^{-\ell_i} + \sum_{i=n+1}^{n+t} e^{-\ell_i} = 1 \qquad \text{(A. 8)}$$

and therefore B is greater than 1 if there are to be non-zero terminal probabilities. The total number of terminal nodes at distance x or less from the origin is now

$$r = \sum_{i=n+1}^{n+t} R(x - \ell_i) \quad . \qquad \text{(A. 9)}$$

Using the smoothed approximation to R, and noting that $x = \ell n \frac{1}{p_r}$, the desired rank probability relationship is approximately

$$r \approx \sum_{i=n+1}^{n+t} e^{-\frac{\ell_i}{B}} \frac{1}{p_r^{1/B}}$$

or

$$p_r \approx \frac{C}{r^B} \quad \text{where} \quad C = \left(\sum_{i=n+1}^{n+t} e^{-\frac{\ell_i}{B}}\right)^B \qquad \text{(A. 10)}$$

A better feeling for the value B may be obtained in the special case where all p_i are equal (to $\frac{1}{n+t}$). In this case

$$B = \frac{\ln(n+t)}{\ln n} \quad . \qquad (A.11)$$

A.4 Generating Graph for the Binomial Distribution

As a contrast to the tree just discussed, consider a generating graph such as shown in Fig. 18. Here all terminal nodes are at the lowest level, and there are many reentrant paths. It can be seen that the number of paths to the i^{th} node at the n^{th} level is $\binom{n}{i}$, and that if the downward choices are made with equal probability, the terminal probabilities are

$$\frac{1}{2^n}\binom{n}{i} \quad . \qquad (A.12)$$

This is maximum for $i=n/2$, and it decreases towards the side approximately according to a Gaussian distribution

$$p_r \sim d^{-kr^2} \quad . \qquad (A.13)$$

This decreases very rapidly compared to r^{-B}. The reentrant paths cause the rapid decrease because they make it less probable that a given traversal will reach the outermost nodes.

The completely reentrant graph of the type shown in Fig. 18 can be described in terms of either a random walk or a summation of random variables (sideways unit deflections). Thus, this case illustrates the central limit theorem.

A.5 Physical Interpretation of Generating Graph

The original interpretation of the tree by Mandelbrot was the generation of a word of a natural language by adding a letter (n=26) or a word-terminating space (t=1). The reentrant graph of Fig. 19 could represent the random walk followed by a ball falling through Galton's pegboard (an educational aid for demonstrating the Gaussian distribution). Returning to the tree, it might represent the repeated division of a whole into parts, thus explaining Zipf-like distributions of income, sand particle sizes, etc.

More to the point here, the tree might represent a repeat of the production rules of a context free grammar. A simple example is the palindrome generator (Fig. 18):

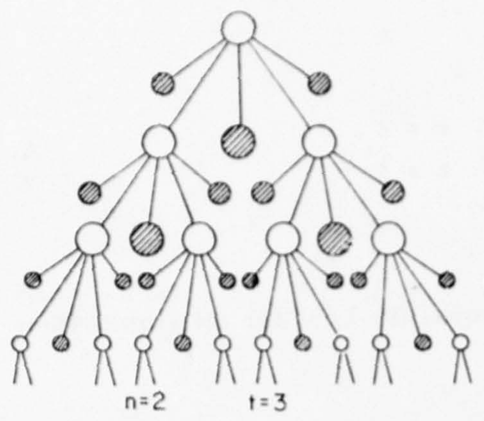

FIG. 18. INFINITE HOMOGENEOUS TREE GIVING A ZIPF DISTRIBUTION.

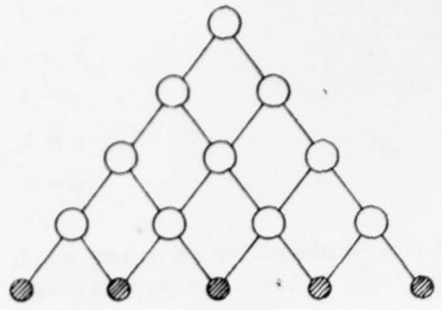

FIG. 19. FINITE REENTRANT-GRAPH GIVING THE BINOMIAL DISTRIBUTION.

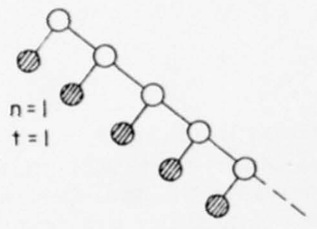

FIG. 20. SPECIAL CASE OF TREE WHICH GIVES GEOMETRIC DISTRIBUTION.

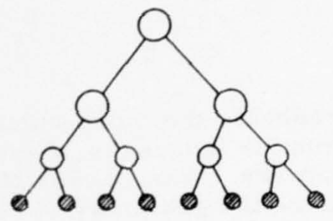

FIG. 21. FINITE TREE GIVEN THE UNIFORM DISTRIBUTION.

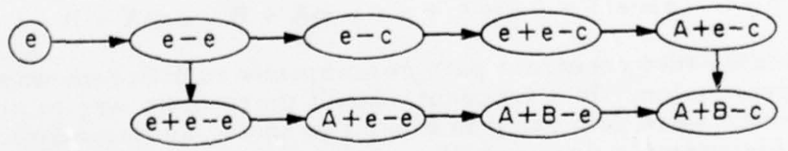

FIG. 22. REENTRANT PATHS IN GENERATING AN ALGEBRAIC EXPRESSION.

$$\begin{array}{ll} \sigma \to 0\,\sigma\,0 & \\ \sigma \to 1\,\sigma & n = 2 \\ \sigma \to 0 & t = 3 \\ \sigma \to 1 & \\ \sigma \to \epsilon & \end{array} \qquad \text{(A. 14)}$$

If the probability of using each rule is equal (to 1/5) for all steps, then B is $\ell n5/\ell n2 = 2.32$ by solving

$$2e^{-\frac{\ell n5}{B}} = 1 \qquad \text{(A. 15)}$$

and the approximate distribution is

$$p_r \approx \frac{C}{r^{2.32}} \quad . \qquad \text{(A. 16)}$$

Probably the most succesful theory of natural languages, and certainly of computer languages, from the mathematical point of view is that started by Chomsky. The Backus Normal Form has become the standard way to formalize the grammar of computer languages. Consider the fragment

$\langle$expression$\rangle$: = expression $\{+|-|/|*\}$ $\langle$expression$\rangle$

$\langle$expression$\rangle$: = A|B|C|D

Let e represent $\langle$expression$\rangle$ and consider the derivations

$$e \Rightarrow e - e \Rightarrow e - C \Rightarrow e + e - C \Rightarrow A + e - C \Rightarrow A + B - C$$

$$e \Rightarrow e - e \Rightarrow e + e - e \Rightarrow A + e - e \Rightarrow A + B - e \Rightarrow A + B - C$$

This illustrates that reentrant path corresponds to different ways to derive the same expression. In a frequency count there is no way to distinguish the A+B-C's which were generated in these two ways, and usually in many others. This example does not illustrate an ambiguous grammar, since only the second derivation is a left most derivation. Since reentrant paths can occur even if the language is inherently unambiguous, the cases between the tree and the Galton board are of paramount interest in studying the statistics of the elements of a programming language.

A. 6 Remark on "Least Effort"

Several authors have explained Zipf's law as a result of a information-maximizing or cost-minimizing process, but this seems to either involve circular reason, or to beg the question. The argument may be simplified to something like this: let c_i be the cost associated with the i^{th} word, perhaps representing its length. If the c_i are subject to constraints such as requiring that no word begin another word, Shannon has shown that

$$H = \sum_i p_1 \log \frac{1}{p_i} \leq \sum_i p_i c_i = C \quad . \tag{A. 17}$$

If we wish to minimize C with fixed H the optimum value of c_i is

$$c_i = \log \frac{1}{p_i} \quad . \tag{A. 18}$$

The same relation should hold if we wish to maximize H with fixed C, or to maximize H/C. <u>If</u> somehow it is known that

$$c_i = \log k i \tag{A. 19}$$

then p_i follows Zipf's law. This merely transfers the problem to explaining why c_i varies logarithmically, and in fact this is usually done by a generating tree such as described above.

References

1. "The Psycho-biology of Language: An Introduction to Dynamic Philology," G. K. Zipf, First Edition 1935 by Houghton Miffin Co., First M.I.T. Press paperback edition, 1965.

2. "National Unity and Disunity," G. K. Zipf, 1941.

3. "Human Behavior and the Principle of Least Effort," G. K. Zipf, Addison-Wesley, 1949.

4. "Language and Communication," G. A. Miller, McGraw-Hill, 1951.

5. "On Human Communication," C. Cherry, M. I. T. Press, Second Edition, 1970.

6. "The Estimation of Probabilities: An Essay on Bayesian Methods," I. J. Good, M. I. T. Press, 1965.

7. "System Engineering," H. Goode and R. Machol, McGraw-Hill, 1957.

8. "The PL/I Machine: An Introduction to Programming," E. Neuhold and H. Lawson, Jr., Addison-Wesley, 1971.

9. "On Recurrent Noise Limiting Coding," B. Mandelbrot, Proceedings of the Symposium on Information Networks, p. 205, Polytechnic Press, Brooklyn, NY, 1954.

10. "On the Theory of Word Frequencies and on Related Markovian Models of Discourse," B. Mandelbrot, Proceedings of Symposia in Applied Mathematics, Vol. XII, p. 190, American Mathematical Society, Providence, RI, 1961.

11. "Study of General Digital Codes with Emphasis on Signal Compression," A. Laemmel, Polytechnic Institute of Brooklyn, Report No. PIBEP-73-125, Farmingdale, NY.

12. "Human Memory and the Storage of Information," G. Miller, I. R. E. Transactions on Information Theory, Vol. IT-2, No. 3, pp. 129-127, 1956.

13. "Study of the Application of Coding Theory," A. Laemmel and B. Rudner, PIBEP-69-034, Polytechnic Institute of Brooklyn, June 1969, p. 4-2.

14. "A Treatise on Advanced Calculus," P. Franklin, Dover Publications, New York, Dover Edition, 1964.

15. Symbols, Signals and Systems in Noise, J. R. Pierce, Harper and Rose, 1965.

References (continued)

16. "A Theory of Word-Frequency Distribution," A. Parker-Rhodes and J. Joyce, Nature, Vol. 178, p. 1308, Dec. 8, 1956.

17. "Software Physics: Basic Principles," M. Halstead, IBM Research Report, RJ1582, IBM Research, Yorktown Heights, NY, May 1975.

18. "An Emperical Study of FORTRAN Programs," D. E. Knuth, Stanford University Computer Science Department Report No. CS-186, 1970.

19. "A Programming Theory Based on Communication Theory, Linguistics, and Switching Theory," A. E. Laemmel, unpublished memo.

20. "Software Engineering," M. L. Shooman, Notes for course ES 909, Polytechnic Institute of New York.

21. "Probabilistic Models for Software Reliability Prediction," M. L. Shooman, p. 495, Statistical Computer Performance Evaluation, W. Freiberger, Ed., Academic Press, New York, 1972.

22. "Three Models for the Description of Language," N. Chomsky, IRE Trans. on Inform. Theory, Vol. IT-2, 1956, pp 113-124 .

23. "The Syntax and Semantics of the Proposed International Algebraic Language," C. Backus, UNESCO conf. on Info. Proc., Paris, 1959, pp. 125-132.